STiMuL: a Software for Modeling Steady-State Temperature in Multilayers - Description and user manual

Pierre Michaud

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

STiMuL : a Software for Modeling Steady-State Temperature in Multilayers - Description and user manual

Pierre Michaud

N° 0385

Avril 2010

ISSN 0249-0803 ISRN INRIA/RT--0385--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

STiMuL : a Software for Modeling Steady-State Temperature in Multilayers - Description and user manual

Pierre Michaud

Thème : Architecture et compilation
Équipes-Projets ALF

Rapport technique n° 0385 — Avril 2010 — 20 pages

Abstract:

This report describes STiMuL, a model and software for steady-state temperature in multilayers. STiMuL is primarily intended for modeling steady-state heat conduction in microprocessors and their packaging. STiMuL can be used to model temperature in 2D and 3D circuits from 2D power density maps provided by the user. It can also be used to obtain power density maps from temperature maps. The STiMuL software is publicly available. This report describes the theoretical model and its software implementation. It explains how to use the software and provides some examples of utilization.

Key-words: microprocessor, thermal model, temperature, steady-state, heat conduction, power density

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

STiMuL : un logiciel de modélisation de la température en régime permanent dans les multicouches - Description et manuel utilisateur

Résumé : Ce rapport décrit STiMuL, un modèle et un logiciel pour la température en régime permanent dans les multicouches. STiMuL est destiné principalement à la modélisation de la conduction de chaleur en régime permanent dans les microprocesseurs. STiMuL peut être utilisé pour modéliser la température dans les circuits 2D et 3D à partir de densités de puissance fournies par l'utilisateur. Il peut aussi être utilisé pour obtenir la densité de puissance à partir de la température. Le logiciel STiMuL est un logiciel libre. Ce rapport décrit le modèle théorique et sa mise en oeuvre logicielle. Il explique comment utiliser le logiciel et donne quelques exemples d'utilisation.

Mots-clés : microprocesseur, modèle thermique, température, régime permanent, conduction de la chaleur, densité de puissance

1 Introduction

STiMuL is a model and software for steady-state temperature in multilayers. It is primarily intended for modeling steady-state heat conduction in microprocessors and their packaging. STiMuL can be used to model steady temperature in 2D and 3D circuits from 2D power density maps provided by the user. It can also be used to obtain power density maps from temperature maps. The STiMuL software is publicly available.

Section 2 describes the theoretical model (which is based on previous work [9]) and its implementation. Section 3 describes the structures and functions of the STiMuL software. Finally, Section 4 gives some examples of utilization.

2 Solving Laplace equation in multilayers

2.1 Analytical solution

The usual way to model heat conduction is to solve a partial differential equation (PDE) with boundary conditions (BCs), i.e., a boundary value problem (BVP).

For the case of steady-state heat conduction in a homogeneous isotropic solid with thermal conductivity independent of temperature and with no heat source inside the solid, temperature $T(x, y, z)$ in the solid verifies Laplace equation [6] :

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} = 0 \tag{1}$$

where x,y,z are cartesian coordinates. Some materials, like silicon, have a thermal conductivity which varies significantly with temperature, which makes the PDE non linear. In this case, it is usual to approximate the problem by taking a fixed thermal conductivity corresponding to the temperature range of the physical problem, so that linear equation (1) applies (other methods are sometimes possible, like Kirchhoff's transformation [6, 5]).

If the solid is heterogeneous but consists of several homogeneous regions, we must solve Laplace equation in each region, with boundary conditions (BCs) on the surfaces of separation between regions (most often, conservation of heat flux normal to the surface and continuity of temperature, but other conditions are possible).

There are several possible methods for solving BVPs that can be applied to heat conduction problems. The most general methods are finite difference (FDM) and finite element methods (FEM). These methods are generally computation intensive because accurate modeling of 3-dimensional heat conduction in regions of high heat flux requires a large number of nodes.

If the shape of the solid is simple, it is sometimes possible to use analytical methods [6, 19], i.e., to search for an explicit solution to the BVP. When they apply, analytical methods are often faster than FDM and FEM. Perhaps more importantly, analytical methods are relatively easy to implement and are easy to use.

There is a particular case which can be used to model heat conduction in integrated circuits : the *multilayer*. A example of multilayer is depicted in Figure 1. A multilayer consists of several layers of possibly different materials, each layer being a cuboid. All layers have the same dimension in the x and y directions (cf. Figure 1). The BCs on the surfaces perpendicular to the z axis can be "complex" ones, e.g., prescribed temperature $T(x, y)$ or prescribed heat flux $\vec{q}(x, y) \cdot \vec{u}_z$ (where $\vec{q}$ is the heat flux vector and $\vec{u}_z$ is the unit vector $(0, 0, 1)$). For the analytical methods to apply, BCs on the lateral sides (that is, surfaces parallel to the z axis) must be "simple" ones, e.g., uniform temperature or adiabatic.

When modeling heat conduction in integrated circuits and their packaging, it is usual to neglect the heat escaping through lateral sides, hence we assume adiabatic BCs on lateral sides :

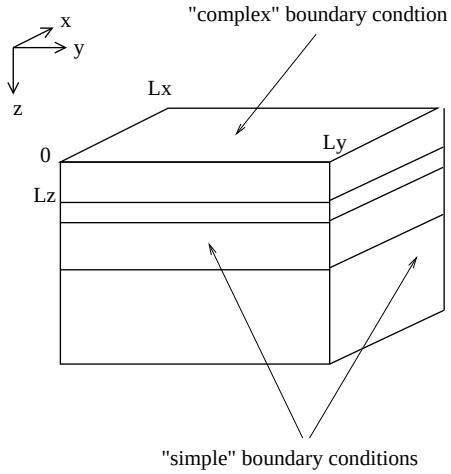

Figure 1: Heat conduction in a multilayer can be solved with analytical methods.

$$\begin{aligned}\frac{\partial T}{\partial x}(0,y,z) &= \frac{\partial T}{\partial x}(L_x,y,z) = 0 \\ \frac{\partial T}{\partial y}(x,0,z) &= \frac{\partial T}{\partial y}(x,L_y,z) = 0\end{aligned} \tag{2}$$

where L_x and L_y are the layers dimensions in the x and y directions respectively (see Figure 1).

Many researchers and engineers have used multilayers and analytical methods to model temperature in integrated circuits and their packaging, e.g., [10, 14, 4, 16, 15, 18, 7, 3, 9, 8, 5, 20, 23, 21, 24, 25, 26, 22] (the list is not exhaustive).

There are several possible methods for obtaining an explicit analytical solution to the multilayer BVP. We used the integral transform method [19, 17]. For a rectangular shape and adiabatic BCs on lateral sides, the appropriate transform is a double cosine transform :

$$\widetilde{T}_{n,m}(z) = \int_0^{L_x}\int_0^{L_y} T(x,y,z)\cos(n\frac{x}{L_x}\pi)\cos(m\frac{y}{L_y}\pi)\,dxdy \tag{3}$$

The transformed temperature $\widetilde{T}$ is in m^2K. The inverse transform involves an infinite summation on n and m integers [19, 17]. It should be noted that

$$\int_0^{L_x}\frac{\partial^2 T}{\partial x^2}(x,y,z)\cos(n\frac{x}{L_x}\pi)\,dx = -\Big(\frac{n\pi}{L_x}\Big)^2\int_0^{L_x} T(x,y,z)\cos(n\frac{x}{L_x}\pi)\,dx$$

(integrate by parts twice and use BCs (2)). Consequently, the transform of $\frac{\partial^2 T}{\partial x^2}$ is

$$\int_0^{L_x}\int_0^{L_y}\frac{\partial^2 T}{\partial x^2}(x,y,z)\cos(n\frac{x}{L_x}\pi)\cos(m\frac{y}{L_y}\pi)\,dxdy = -\Big(\frac{n\pi}{L_x}\Big)^2\widetilde{T}_{n,m}(z) \tag{4}$$

Similarly, the transform of $\frac{\partial^2 T}{\partial y^2}$ is

$$\int_0^{L_x}\int_0^{L_y}\frac{\partial^2 T}{\partial y^2}(x,y,z)\cos(n\frac{x}{L_x}\pi)\cos(m\frac{y}{L_y}\pi)\,dxdy = -\Big(\frac{m\pi}{L_y}\Big)^2\widetilde{T}_{n,m}(z) \tag{5}$$

Finally, the transform of $\frac{\partial^2 T}{\partial z^2}$ is

$$\int_0^{L_x}\int_0^{L_y} \frac{\partial^2 T}{\partial z^2}(x,y,z)\cos(n\frac{x}{L_x}\pi)\cos(m\frac{y}{L_y}\pi)\,dxdy = \frac{\partial^2}{\partial z^2}\widetilde{T}_{n,m}(z) \tag{6}$$

Summing equations (4), (5) and (6), Laplace equation (1) becomes

$$\frac{\partial^2}{\partial z^2}\widetilde{T}_{n,m}(z) = \gamma_{n,m}^2 \times \widetilde{T}_{n,m}(z) \tag{7}$$

with $\gamma_{n,m}$ defined as

$$\gamma_{n,m} = \pi\sqrt{\left(\frac{n}{L_x}\right)^2 + \left(\frac{m}{L_y}\right)^2} \tag{8}$$

Like Laplace equation (1), the ordinary differential equation (7) holds within a single homogeneous layer. Its general solution is

$$\widetilde{T}_{n,m}(z) = A_{n,m} \times e^{\gamma_{n,m}z} + B_{n,m} \times e^{-\gamma_{n,m}z} \tag{9}$$

The unknowns $A_{n,m}$ and $B_{n,m}$ are uniquely determined by specifying BCs on the planes delimiting the layer (perpendicular to the z axis). These two planes are at $z = z_0$ and $z = z_1 = z_0 + L_z$, where L_z is the layer thickness.

We consider two types of BCs : prescribed temperature and prescribed heat flux in the z direction. The heat flux vector is $\vec{q} = -k\vec{\nabla}T$ according to Fourier's law, where k is the material thermal conductivity (W/mK). The heat flux in the z direction is

$$Q = \vec{q}\cdot\vec{u}_z = -k\frac{\partial T}{\partial z}$$

Q is in W/m^2. The double cosine transform of Q is

$$\widetilde{Q}_{n,m}(z) = -k\frac{\widetilde{T}}{\partial z} = -k\gamma_{n,m}\Big[A_{n,m} \times e^{\gamma_{n,m}z} - B_{n,m} \times e^{-\gamma_{n,m}z}\Big] \tag{10}$$

The transformed heat flux $\widetilde{Q}$ is in W. As emphasized in [9, 17, 23], it is convenient to view the relation between $(\widetilde{T}_{n,m}(z_0), \widetilde{Q}_{n,m}(z_0))$ and $(\widetilde{T}_{n,m}(z_1), \widetilde{Q}_{n,m}(z_1))$ as a two-port network (aka *quadrupole* [17]). Note that $A_{n,m}$ and $B_{n,m}$ can be written

$$\begin{aligned} A_{n,m} &= \frac{1}{2}\Big[\widetilde{T}_{n,m}(z_0) - \frac{\widetilde{Q}_{n,m}(z_0)}{k\gamma_{n,m}}\Big]e^{-\gamma_{n,m}z_0} \\ B_{n,m} &= \frac{1}{2}\Big[\widetilde{T}_{n,m}(z_0) + \frac{\widetilde{Q}_{n,m}(z_0)}{k\gamma_{n,m}}\Big]e^{\gamma_{n,m}z_0} \end{aligned}$$

In matrix form, we have

$$\begin{pmatrix} \widetilde{T}_{n,m}(z_1) \\ \widetilde{Q}_{n,m}(z_1) \end{pmatrix} = \begin{pmatrix} \cosh(\gamma_{n,m}L_z) & -\frac{1}{k\gamma_{n,m}}\sinh(\gamma_{n,m}L_z) \\ -k\gamma_{n,m}\sinh(\gamma_{n,m}L_z) & \cosh(\gamma_{n,m}L_z) \end{pmatrix} \begin{pmatrix} \widetilde{T}_{n,m}(z_0) \\ \widetilde{Q}_{n,m}(z_0) \end{pmatrix}$$

This corresponds to the "T" network depicted on Figure 2, where $\widetilde{T}$ and $\widetilde{Q}$ are equivalent to voltage and current respectively. The thermal impedances $H_{n,m}$ and $V_{n,m}$ of the "T" network are

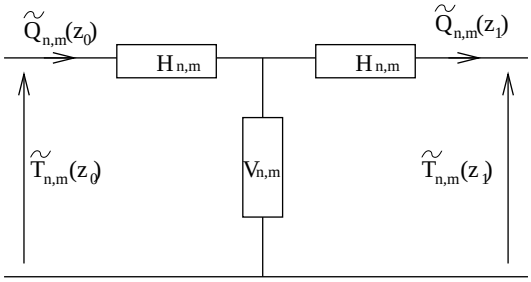

Figure 2: Network representation for a homogeneous layer : transformed temperature and heat flux are equivalent to voltage and current respectively.

$$
\begin{aligned}
H_{n,m} &= \frac{\cosh(\gamma_{n,m} L_z) - 1}{k\gamma_{n,m} \sinh(\gamma_{n,m} L_z)} \\
V_{n,m} &= \frac{1}{k\gamma_{n,m} \sinh(\gamma_{n,m} L_z)}
\end{aligned}
$$

Figure 3 shows an example of multilayer and its network equivalent. Power sources are planar, i.e., infinitely thin (W/m^2). The double cosine transform of a source $q(x, y)$ is equivalent to a current source in the network representation. On this example, h_0, h_1 and h_2 are thermal conductances in W/m^2K. Thermal conductances can be used to model heat exchange with the ambient medium or imperfect contact between layers. In the network representation, a thermal conductance h in W/m^2K is represented by an electrical resistance $1/h$ independent of n and m. It should be noted that a thermal conductance can be viewed as an infinitely thin material layer. A very thin layer of thickness L_z (small) and thermal conductivity k can be approximated by a thermal conductance

$$h \approx \frac{k}{L_z} \tag{11}$$

We can obtain the heat flux or the temperature at any interface simply by solving an electrical network.

2.2 Double cosine transform and its inverse

The solution we obtain in the transformed space (n, m) is the exact solution to the BVP. As mentioned previously, the inverse of the double cosine transform (3) involves an infinite summation on n and m integers.

In order to decrease the computation time, the transform and its inverse can be implemented through discrete cosine transforms, for which efficient algorithms are known.

Let us consider $W(x, y)$, which represents either temperature $T(x, y)$, heat flux $Q(x, y)$, or power sources $q(x, y)$. The transform of W is

$$\widetilde{W}_{n,m} = \int_0^{L_x} \int_0^{L_y} W(x, y) \cos(n\frac{x}{L_x}\pi) \cos(m\frac{y}{L_y}\pi)\, dxdy$$

We discretize the plane $[0, L_x] \times [0, L_y]$ into a regular grid $\{0, \cdots, N_x - 1\} \times \{0, \cdots, N_y - 1\}$ with N_x and N_y integers. Each grid element is a rectangle of size $(L_x/N_x) \times (L_y/N_y)$. If the grid element is small enough, $W(x, y)$ is approximately uniform in each grid element and $\widetilde{W}_{n,m}$ is approximately equal to

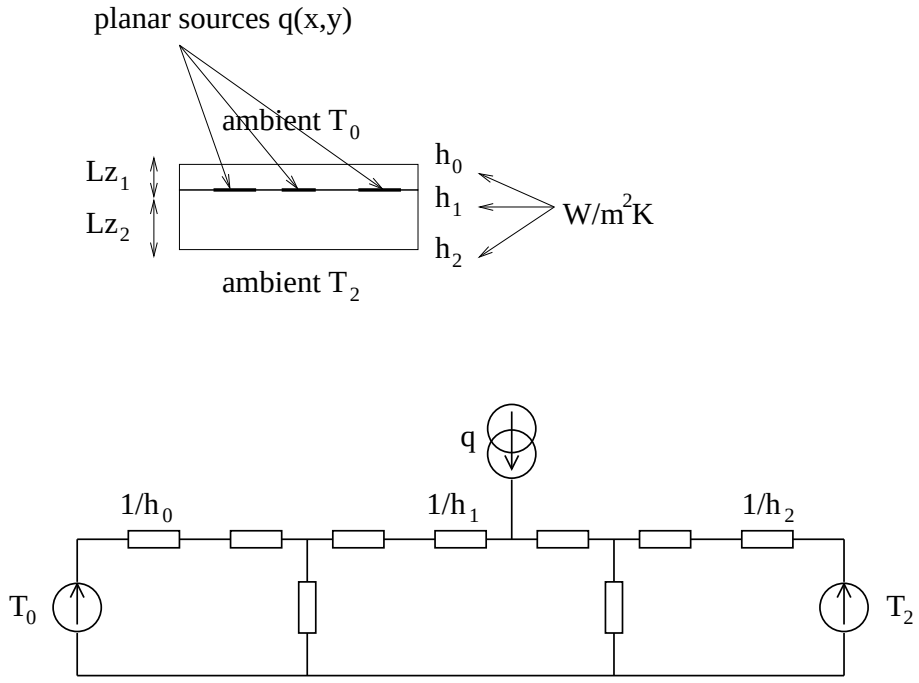

Figure 3: Example of multilayer and its network equivalent. Planar power sources are in W/m^2.

$$\begin{aligned}\widetilde{W}'_{n,m} &= \sum_{i=0}^{N_x-1}\sum_{j=0}^{N_y-1} W(i\frac{L_x}{N_x}, j\frac{L_y}{N_y}) \int_{i\frac{L_x}{N_x}}^{(i+1)\frac{L_x}{N_x}} \int_{j\frac{L_y}{N_y}}^{(j+1)\frac{L_y}{N_y}} \cos(n\frac{x}{L_x}\pi)\cos(m\frac{y}{L_y}\pi)\,dxdy \\ &= C_{n,m}\sum_{i=0}^{N_x-1}\sum_{j=0}^{N_y-1} W(i\frac{L_x}{N_x}, j\frac{L_y}{N_y})\cos(\frac{n\pi}{N_x}(i+\frac{1}{2}))\cos(\frac{m\pi}{N_y}(j+\frac{1}{2})) \qquad (12)\end{aligned}$$

with $C_{n,m}$ defined as

$$C_{n,m} = \frac{L_x L_y}{N_x N_y}\,\mathrm{sinc}(\frac{n\pi}{2N_x})\,\mathrm{sinc}(\frac{m\pi}{2N_y})$$

where $\mathrm{sinc}(x) = \sin(x)/x$ for $x \neq 0$ and $\mathrm{sinc}(0) = 1$. That is, $\widetilde{W}'_{n,m}/C_{n,m}$ is the type-II double discrete cosine transform (DCT) of W. W can be recovered from $\widetilde{W}'$ by a type-III double DCT :

$$W(i\frac{L_x}{N_x}, j\frac{L_y}{N_y}) = \frac{1}{N_x N_y}\sum_{n=0}^{N_x-1}\sum_{m=0}^{N_y-1}\sigma_n\sigma_m\frac{\widetilde{W}'_{n,m}}{C_{n,m}}\cos(\frac{n\pi}{N_x}(i+\frac{1}{2}))\cos(\frac{m\pi}{N_y}(j+\frac{1}{2})) \qquad (13)$$

with $\sigma_0 = 1$ and $\sigma_n = 2$ for $n > 0$. In practice, DCTs are implemented as Fast Fourier Transforms (FFT). If we take N_x and N_y sufficiently large, we can obtain an arbitrarily accurate numerical evaluation of the multilayer BVP solution.

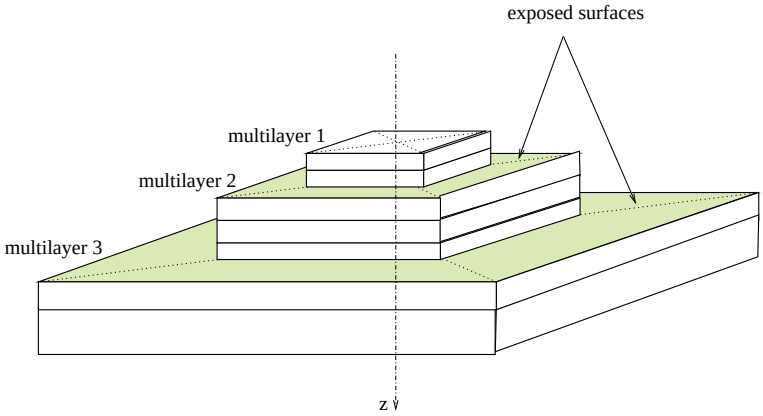

Figure 4: Example of 3-multilayer stack. We assume the (multi)layers are centered, i.e., their centers lie on a straight line parallel to the z axis.

2.3 Multilayer stack

The analytical method described in Section 2.1 applies only to layers having the same L_x and L_y dimensions. This method does not apply to structures like the one depicted on Figure 4. We call the kind of structure depicted on Figure 4 a *multilayer stack*, or *stack* for short. The analytical method does not apply here because the transform (3) depends on L_x and L_y.

Nevertheless, as emphasized in [9], it is possible to obtain an accurate solution by using successive approximations. The method we have implemented is very similar to the one described in [9]. During an iteration, each multilayer is solved assuming approximate BCs. Solutions obtained during an iteration are used to refine the BCs for the next iteration, and so on.

To simplify the implementation, we made the following assumptions. First, we assume that all multilayers are centered, i.e., that their centers lie on a straight line parallel to the z axis. This assumption is not essential, but is sufficient for modeling processors. Second, we assume that L_x and L_y increase with z, i.e., multilayer 2 is wider than multilayer 1, multilayer 3 is wider than multilayer 2, and so on (cf. Figure 4). Third, on the *exposed surfaces*, i.e., the surfaces that are exposed to the ambient medium *because* adjacent multilayers have different widths (cf. Figure 4), we assume a convective boundary condition :

$$Q(x,y) = h_n \times (T_n^0 - T_n(x,y)) \tag{14}$$

where $Q(x,y)$ is the heat flux in the z direction, $n \geq 2$ is the multilayer rank, T_n^0 is the ambient temperature in this region (assumed uniform all over the exposed surface), $T_n(x,y)$ is the temperature on the exposed multilayer surface, and h_n is a thermal conductance (aka heat transfer coefficient) in W/m^2K.

The method we have implemented works as follows. The BC on the multilayer side in contact with a smaller multilayer (side 0) is a flux BC. The BC on the multilayer side in contact with a bigger multilayer (side 1) is a temperature BC.

The iterative algorithm computes temperature on side 0 of each multilayer. Initially, this temperature is set to a constant value. Temperature on side 0 of multilayer n determines the BC on side 1 of multilayer $n-1$ and, through formula (14), the heat flux on the exposed surface of multilayer n. During an iteration, solving multilayer $n-1$ provides the heat flux in the non-exposed zone of side 0 of multilayer n. This way, we can solve successively multilayer 1, multilayer 2, multilayer 3, and so on. We call this an *iteration*. On iteration j, the old temperature $T_{j-1}(x,y)$ on side 0 is replaced with the newly computed temperature $T_j(x,y) = T(x,y)$.

In the good case, only a few iterations are necessary to converge to the solution. However, as noted in [9], there are cases where, instead of converging, temperature diverges. This happens in particular when the thermal resistance of multilayer n is larger than the total thermal resistance from multilayer 1 to multilayer $n-1$. The solution proposed in [9] is as follows. Instead of replacing the old temperature $T_{j-1}(x,y)$ of side 0 of multilayer n with the new temperature $T(x,y)$, we have

$$T_j(x,y) = \alpha_n T(x,y) + (1-\alpha_n) T_{j-1}(x,y)$$

with $\alpha_n \in]0,1]$. The value of α_n must be small enough to ensure convergence. However, as noted in [9], the convergence time can be much longer for smaller α_n values. We implemented a heuristic that tries to find good values for α_n automatically. We found that, sometimes, different multilayers in the same stack require different α_n values.

3 The STiMuL software

The StiMuL software is publicly available [2]. It is implemented as a set of C functions. It uses the FFTW library [11, 1].

3.1 Grid functions

The basic structure of the STiMuL software is the *grid*, which is a 2-dimensional array of floating-point values defined as follows :

```
typedef double stml_realtype;

typedef struct {
  int nx;
  int ny;
  stml_realtype v[STML_GRIDMAX][STML_GRIDMAX];
} stml_grid;
```

The software provides several functions for manipulating grids. The following function function allocates a grid of size $nx \times ny$ floating-point values and returns a pointer to it :

```
stml_grid * stml_grid_new(int nx,int ny);
```

The grid can be freed with the following function :

```
void stml_grid_free(stml_grid *g);
```

The following function sets grid g uniformly to value val :

```
void stml_grid_set(stml_grid *g,double val);
```

The following function sets a part of grid g to value val. The part is a rectangle defined by the coordinates $(i1, j1)$ and $(i2, j2)$ of two vertices defining a diagonal of the rectangle (the rectangle sides are parallel to the grid ones) :

```
void stml_grid_rectangle(stml_grid *g,int i1,int j1,int i2,int j2,double val);
```

The following function sets grid g dimensions to nx and ny and resets its value uniformly to zero.

```
void stml_grid_reset(stml_grid *g,int nx,int ny);
```

A grid g can also be viewed as a grid of $nx \times ny$ square cells of unit length. In this case, the value $g.v[i][j]$ represents the value at the point of coordinates $(i+\frac{1}{2}, j+\frac{1}{2})$ in the grid, i.e., the value at the center of the cell. The following function returns the value of grid g at the point of coordinates (x,y), where $x \in [0, nx]$ and $y \in [0, ny]$ are floating-point values (the value is obtained by linear interpolation) :

```
double stml_grid_read(stml_grid *g,double x,double y);
```

The following functions return respectively the average, maximum and minimum value of grid g :

```
double stml_grid_average(stml_grid *g);
double stml_grid_max(stml_grid *g);
double stml_grid_min(stml_grid *g);
```

The following function prints grid g on the standard output :

```
void stml_grid_print(stml_grid *g);
```

3.2 Multilayer functions

The $stml\_multilayer$ type is a structure representing a multilayer, as depicted in Figure 1.

The following function allocates a multilayer and returns a pointer to it :

```
stml_multilayer * stml_multilayer_new(double lx,double ly,int nx,int ny);
```

Parameters lx and ly are the dimensions in meters in the x and y directions respectively (see Figure 1). Parameter nx and ny are the dimensions of the grids associated with the multilayer. The larger nx and ny, the more accurate the solution. The following function frees a multilayer p allocated with $stml\_multilayer\_new$:

```
void stml_multilayer_free(stml_multilayer *p);
```

3.2.1 Defining active layers

Active layers can be of two sorts : temperature or source. The following function defines an isothermal plane of temperature t (in °C) in multilayer p :

```
void stml_multilayer_isotherm(stml_multilayer *p,double t,const char *name);
```

where $name$ is a string for identifying the layer. This function can be used to set a temperature boundary condition. The following function can also be used to define a temperature boundary condition but is more general :

```
void stml_multilayer_temperature(stml_multilayer *p,stml_grid *g,const char *name);
```

where, instead of being uniform, temperature is specified with a grid g whose dimensions nx and ny must match the multilayer ones. Functions $stml\_multilayer\_isotherm$ and $stml\_multilayer\_temperature$ can be used only to define outside boundary conditions, i.e., the first and/or the last layer in a multilayer. They cannot be used inside the multilayer.

The following function defines a heat source plane :

```
void stml_multilayer_sources(stml_multilayer *p,stml_grid *g,const char *name);
```

where the planar sources are specified with a grid g representing a 2-dimensional power density in watt per meter square (parameters nx and ny of g must match those of p). Function $stml\_multilayer\_sources$ can be used to define a heat flux boundary condition (i.e., when sources lie on the outside boundary). It can also be used to define heat sources inside the multilayer. If parameter g is set to NULL, there is no heat generation on the layer. This is useful when one wants to obtain temperature on a plane with no heat sources.

Each active layer is identified with a string $name$, and different active layer must have different names.

3.2.2 Defining passive layers

Passive layers can be of two sorts : slab or interface. The following function defines a slab of thickness lz in meters (see Figure 1) and whose material has a thermal conductivity k in watt per meter and per kelvin :

```
void stml_multilayer_slab(stml_multilayer *p,double lz,double k)
```

The following function defines an infinitely thin layer of conductance h in watt per meter square and per kelvin (Formula 11) :

```
void stml_multilayer_interface(stml_multilayer *p,double h);
```

Such infinitely thin layer can be used to simulate heat transfer by convection on the outside boundary. It can also be used to simulate an imperfect contact between two slabs.

3.2.3 Solving the multilayer

Once all layers of a multilayer have been defined, the following function must be used to finalize the multilayer :

```
void stml_multilayer_finalize(stml_multilayer *p);
```

This function prepares the multilayer for being solved and checks its consistency. In order to be consistent, a multilayer must verify the following conditions :

- The multilayer must have at least 3 layers
- The first and last layer must be active layers (as they define boundary conditions)
- The first or the last layer must be a temperature layer (otherwise there is not a unique solution)
- A temperature layer can only be the first or the last layer, or both (a temperature layer defines a boundary condition).
- Active layers must be separated by one or several passive layers

Once the multilayer p is finalized, it can be solved with the following function :

```
stml_multilayer_solve(stml_multilayer *p, const char *name, stml_grid *g);
```

where $name$ is the active layer for which we search a solution, and g is the grid in which the solution will be written (parameters nx and ny of the grid must match those of p).

If the active layer is a temperature layer, the solution obtained in g is the heat flux in W/m^2 on that layer. If the active layer is a source layer, the solution obtained in g is the temperature in °C on that layer.

Once a multilayer has been finalized and solved ($stml\_multilayer\_solve$ may be called several times), we may want to change some of its characteristics. In general, this requires to define a new multilayer. But if we just want to change the values of an active layer, it is not necessary to define a new multilayer. The following function may be called instead :

```
void stml_multilayer_modify(stml_multilayer *p,const char *name,stml_grid *g);
```

where $name$ is the active layer name and g is the new active layer value.

3.3 Multilayer-stack functions

A multilayer-stack, or *stack* for short, consists of several multilayers, as illustrated in Figure 4. The BCs on the outside stack surfaces are all temperature BCs.

A stack is represented in the STiMuL software by the type $stml\_stack$. The following function allocates a stack and returns a pointer to it :

```
stml_stack * stml_stack_new();
```

The stack can eventually be freed with the following function :

```
void stml_stack_free(stml_stack *s);
```

The first multilayer p in a stack s is the multilayer with the smallest lx and ly values. It is included in the stack with the following stack initialization function :

```
void stml_stack_start(stml_stack *s,
                      double lx,double ly,
                      int nx,int ny,
                      stml_grid *g,double t,
                      const char *name);
```

where lx and ly are the dimensions of the first multilayer and nx and ny are its grid dimensions. Parameter $name$ is the name of the first multilayer. Grid g is used to set the temperature BC for the first multilayer (i.e., the BC on the top of the stack, see Figure 4). Temperatures are in degrees Celsius. If g is set to NULL, temperature is uniform and equal to parameter t.

Subsequent multilayers can be added to the stack with the following function

```
stml_stack_add(stml_stack *s,
               double lx,double ly,
               int nx,int ny,
               double h,double t,
               const char *name);
```

where lx and ly are the multilayer dimensions, nx and ny its grid dimensions, h is the heat transfer coefficient (in W/m^2K) of the multilayer exposed surface (see Figure 4 and Section 2.3) and t is the temperature (in °C) of the ambient medium at that surface. Parameters h and t correspond to h_n and T_n^0 in formula (14).

The only contraints when adding a multilayer to a stack having already one or several multilayers is that the dimensions lx and ly of the multilayer must be greater than the dimensions of the previous multilayer in the stack, and the name of the multilayer must be different from that of the other multilayers in the stack.

The following function adds an interface of conductance h (in W/m^2K) to multilayer $name$:

```
stml_stack_interface(stml_stack *s,const char *name,double h);
```

The following function adds a slab of thickness lz (in m) and thermal conductivity k (in W/mK) to multilayer $name$:

```
void stml_stack_slab(stml_stack *s,const char *name,double lz,double k);
```

The following function adds a source layer named $name2$ to multilayer $name1$:

```
void stml_stack_sources(stml_stack *s,const char *name1,
                        stml_grid *g,const char *name2);
```

where g is the power density grid in W/m^2. A constraint is that a source layer must be preceded and followed by a passive layer in the same multilayer (i.e., a slab or an interface). The layer name can be any string but "top" and "bottom", which are reserved names.

Once all the multilayers constituting the stack have been defined, the stack must be finalized with the following function :

```
void stml_stack_finalize(stml_stack *s,stml_grid *g,double t);
```

where g is the temperature grid defining the BC on the bottom side of the stack (see Figure 4). If g is set to NULL, temperature is assumed uniform and equal to t.

Once finalized, the stack is solved with the following function :

```
void stml_stack_solve(stml_stack *s,double eps);
```

Parameter eps is used to control the accuracy of the solution : the smaller eps, the more accurate. For instance, if we want the relative error to be of the order of 10^{-2} (and provided grid dimensions are set accordingly), we should take $eps = 10^{-2}$ or smaller.

Once the stack is solved, the following function allows to read the solutions :

```
void stml_stack_solution(stml_stack *s,const char *name1,
                         const char *name2,stml_grid *g);
```

where $name1$ is the multilayer name, $name2$ is the active layer name in that multilayer, and g is the grid where the solution is written. If $name2$ is set to "top", the solution obtained is that on the first active layer. The first active layer of the first multilayer is a temperature BC, hence the corresponding solution is the heat flux entering the top of the stack. For example, if the first multilayer is named "ML1", the heat flux on the top of the stack is obtained with $stml\_stack\_solution(s,"ML1","top",g)$. For multilayers other than the first one, the first active layer is a flux boundary condition, hence the solution obtained is the temperature on that layer. The last layer in a multilayer is named "bottom", and it is always a temperature boundary condition. For intance, $stml\_stack\_solution(s,"ML","bottom",g)$ permits obtaining the heat flux on the last layer of the multilayer named "ML".

If we want to modify the values of an active layer without redefining a new stack, we can use the following function :

```
void stml_stack_modify(stml_stack *s,const char *name1,
                       const char *name2,stml_grid *g);
```

where $name1$ is the multilayer name, $name2$ is the active layer name in that multilayer, and g is the new value. For instance, if the first multilayer is named "ML1", $stml\_stack\_modify(s,"ML1","top",g)$ changes the temperature BC on the top of the stack.

4 Examples

4.1 A simple multilayer

The example program of Figure 5 models a simple multilayer consisting of a single $1\,cm \times 1\,cm \times 1\,mm$ slab of thermal conductivity $400\,W/mK$. A uniform power density of $1\,W/mm^2$ is applied on the top side of the slab and a uniform temperature of 20°C is applied on the bottom side. The program in Figure 5 gives the temperature on the first side and the heat flux on the other side.

4.2 Example of multilayer stack

Let us assume we want to model a processor and its packaging as depicted on Figure 6. We can model it with the program in Figure 7. We model power dissipation by transistors and wires with two square heat sources lying on the same heat source plane (it should be noted that it is possible with STiMuL to model 3D circuits with several heat source planes). Eventually, we output the temperature on the heat source plane. The corresponding temperature map is shown in Figure 8.

```c
#include <stdio.h>
#include "stimul.h"

// square layer of width 1 cm
#define WX 0.01
#define WY WX

// use grids of 100 x 100
#define NX 100
#define NY NX

int main()
{
  stml_grid *g = stml_grid_new(NX,NY);
  stml_multilayer *p = stml_multilayer_new(WX,WY,NX,NY);

  // first layer defines a heat flux of 1 W/mm^2 (top side)
  stml_grid_set(g,1e6);
  stml_multilayer_sources(p,g,"top side");

  // second layer defines a slab of thickness 1 mm and thermal conductivity 400 W/mK
  stml_multilayer_slab(p,1e-3,400);

  // last layer defines an isothermal plane at 20 degrees Celsius (bottom side)
  stml_multilayer_isotherm(p,20,"bottom side");

  // finalize multilayer
  stml_multilayer_finalize(p);

  // get temperature at the center of the top side
  stml_multilayer_solve(p,"top side",g);
  printf("Temperature = %f\n",stml_grid_read(g,NX/2,NY/2));

  // get heat flux at the center of the bottom side
  stml_multilayer_solve(p,"bottom side",g);
  printf("Heat flux = %f\n",stml_grid_read(g,NX/2,NY/2));

  stml_grid_free(g);
  stml_multilayer_free(p);
}
```

Figure 5: Program of the example 4.1

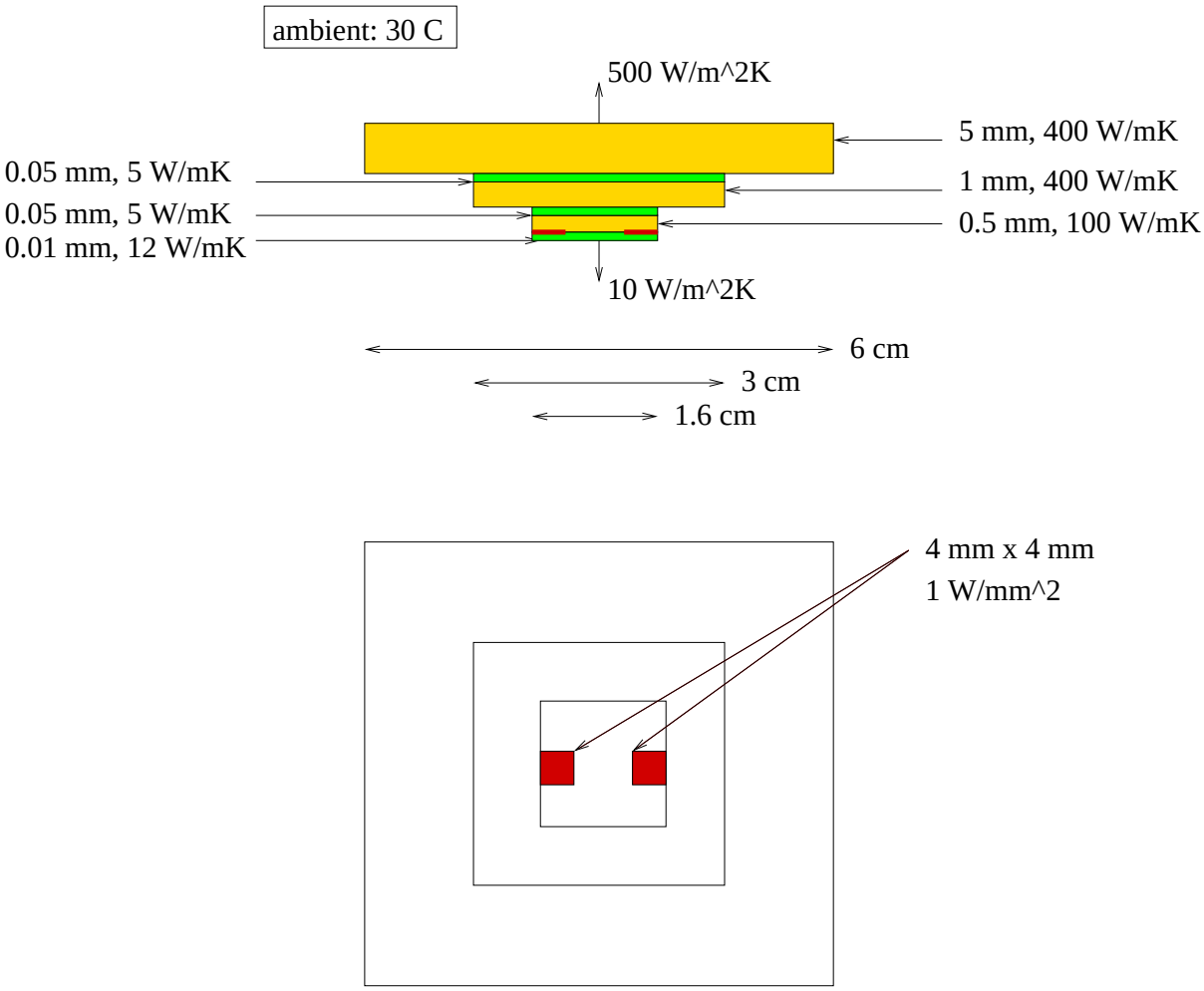

Figure 6: Example 4.2.

```
#include <stdio.h>
#include "stimul.h"

#define AMBIENT 30

// use grids of 160 x 160
#define NX 160
#define NY NX

int main()
{
  stml_stack *s = stml_stack_new();
  stml_grid *g = stml_grid_new(NX,NY);

  // set the two square heat sources in grid g
  stml_grid_set(g,0);
  stml_grid_rectangle(g,0,NX/2-NX/8,NX/4-1,NX/2+NX/8-1,1e6);
  stml_grid_rectangle(g,NX-1,NX/2-NX/8,NX-NX/4,NX/2+NX/8-1,1e6);

  // define 3 multilayers
  stml_stack_start(s,0.016,0.016,NX,NY,NULL,AMBIENT,"ML1");
  stml_stack_add(s,0.03,0.03,NX,NY,100,AMBIENT,"ML2");
  stml_stack_add(s,0.06,0.06,NX,NY,100,AMBIENT,"ML3");
  // first multilayer
  stml_stack_interface(s,"ML1",10);
  stml_stack_slab(s,"ML1",1e-5,12);
  stml_stack_sources(s,"ML1",g,"circuits");
  stml_stack_slab(s,"ML1",5e-4,100);
  stml_stack_slab(s,"ML1",5e-5,5);
  // second multilayer
  stml_stack_slab(s,"ML2",1e-3,400);
  stml_stack_slab(s,"ML2",5e-5,5);
  // third multilayer
  stml_stack_slab(s,"ML3",5e-3,400);
  stml_stack_interface(s,"ML3",500);
  // finalize and solve
  stml_stack_finalize(s,NULL,AMBIENT);
  stml_stack_solve(s,1e-3);
  // output temperature map of the source layer
  stml_stack_solution(s,"ML1","circuits",g);
  stml_grid_print(g);

  stml_grid_free(g);
  stml_stack_free(s);
}
```

Figure 7: Program of the example 4.2

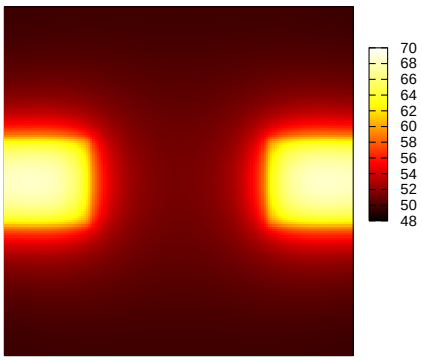

Figure 8: Output of program 7.

4.3 Example: determining power density from temperature

Determining the power density map of a microprocessor executing a given program is not straightforward. Some researchers have shown that it is possible to obtain power density indirectly by infrared thermal imaging [12, 13], i.e., by deriving the power density map from a temperature map. A possible solution (the one used in [12, 13]) is to obtain the thermal resistance matrix, either by measurement or modeling, and solve the system of equations.

STiMuL offers another possible method. The program in Figure 9 shows an example of obtaining a power density map from a temperature map. In this case, the temperature map is used as a boundary condition.

In general, power density is very sensitive to input parameters (temperature, thermal conductivity, etc.) so in practice it is recommended to vary the inputs in order to quantify the error interval.

References

[1] FFTW. http://www.fftw.org.

[2] STiMuL. http://www.irisa.fr/alf/stimul.

[3] J. Albers. An exact recursion relation solution for the steady-state surface temperature of a general multilayer structure. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 18(1):31–38, March 1995.

[4] P. Antognetti, G.R. Bisio, F. Curatelli, and S. Palara. Three-dimensional transient thermal simulation : application to delayed short circuit protection in power IC's. *IEEE Journal of Solid-State Circuits*, SC-15(3):277–281, June 1980.

[5] W. Batty, C.E. Christoffersen, A.J. Panks, S. David, C.M. Snowden, and M.B. Steer. Electro-thermal CAD of power devices and circuits with fully physical time-dependent compact thermal modelling

```
#include <stdio.h>
#include "stimul.h"

#define NX 180
#define NY 230
#define WIDTH 0.1
#define CHIPX 0.00811
#define CHIPY 0.01036
#define AMBIENT 26.112

int main()
{
  int i,j;
  FILE *ft = fopen("inputs/temperature_map","r");
  stml_grid *g = stml_grid_new(NX,NY);
  stml_stack *s = stml_stack_new();

  stml_grid_set(g,0);
  for (j=NY-1; j>=0; j--) {
    for (i=0; i<NX; i++) {
      fscanf(ft,"%lf",&g->v[i][j]);
    }
  }
  fclose(ft);

  stml_stack_start(s,CHIPX,CHIPY,NX,NY,g,0,"ML1");
  stml_stack_add(s,WIDTH,WIDTH,NX,NY,0,0,"ML2");
  stml_stack_slab(s,"ML1",5e-4,129.16);
  stml_stack_interface(s,"ML1",3.33e4);
  stml_stack_slab(s,"ML2",1e-3,400);
  stml_stack_finalize(s,NULL,AMBIENT);
  stml_stack_solve(s,1e-3);
  stml_stack_solution(s,"ML1","top",g);
  stml_grid_print(g);

  stml_grid_free(g);
  stml_stack_free(s);
}
```

Figure 9: Program of the example 4.3 : obtaining power density from temperature.

of complex non linear 3-dimensional systems. *IEEE Transactions on Components and Packaging Technologies*, 24(4):566–590, December 2001.

[6] H.S. Carslaw and J.C. Jaeger. *Conduction of heat in solids*. Oxford University Press, 1959.

[7] D.H. Chien, C.Y. Wang, and C.C. Lee. Temperature solution of five-layer structure with an embedded circular source. In *Proceedings of the IEEE Intersociety Conference on Thermal Phenomena in Electronic Systems (I-THERM)*, 1992.

[8] J.R. Culham, M.M. Yovanovich, and T.F. Lemczyk. Thermal characterization of electronic packages using a three-dimensional Fourier series solution. *Journal of Electronic Packaging*, 122(3):233–239, September 2000.

[9] J.-M. Dorkel, P. Tounsi, and P. Leturcq. Three-dimensional thermal modeling based on the two-port network theory for hybrid or monolithic integrated power circuits. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 19(4):501–507, December 1996.

[10] G.N. Ellison. The effect of some composite structures on the thermal resistance of substrates and integrated circuit chips. *IEEE Transactions on Electron Devices*, ED-20(3):233–238, March 1973.

[11] M. Frigo and S.G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, February 2005.

[12] H.F. Hamann, J. Lacey, A. Weger, and J. Wakil. Spatially-resolved imaging of microprocessor power SIMP : hotspots in microprocessors. In *Proceedings of the 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2006.

[13] H.F. Hamann, A. Weger, J.A. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil. Hotspot-limited microprocessors : direct temperature and power distribution measurements. *IEEE Journal of Solid-State Circuits*, 42(1):56–65, January 2007.

[14] A.G. Kokkas. Thermal analysis of multiple-layer structures. *IEEE Transactions on Electron Devices*, ED-21(11):674–681, November 1974.

[15] C.C. Lee, J. Min, and A.L. Palisoc. An integration algorithm for the temperature solution of the four-layer infinite plate structure. In *Proceedings of the 5th SEMI-THERM Symposium*, 1989.

[16] P. Leturcq, J.-M. Dorkel, A. Napieralski, and E. Lachiver. A new approach to thermal analysis of power devices. *IEEE Transactions on Electron Devices*, ED-34(5):1147–1156, May 1987.

[17] D. Maillet, S. André, J.C. Batsale, A. Degiovanni, and C. Moyne. *Thermal quadrupoles - Solving the heat equation through integral transforms*. Wiley, 2000.

[18] Y.J. Min, A.L. Palisoc, and C.C. Lee. Transient thermal study of semiconductor devices. In *Proceedings of the 6th SEMI-THERM Symposium*, 1990.

[19] M.N. Özişik. *Boundary value problems of heat conduction*. Dover Publications, 1968.

[20] N. Rinaldi. Generalized image method with application to the thermal modeling of power devices and circuits. *IEEE Transactions on Electron Devices*, 49(4):679–686, April 2002.

[21] B. Wang and P. Mazumder. Fast thermal analysis for VLSI circuits via semi-analytical Green's function in multi-layer materials. In *Proceedings of the International Symposium on Circuits and Systems*, 2004.

[22] B. Wang and P. Mazumder. Acelerated chip-level thermal analysis using multilayer Green's function. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):325–344, February 2007.

[23] K. Xiu and M. Ketchen. Thermal modeling of a small extreme power density macro on a high power density microprocessor chip in the presence of realistic packaging and interconnect structures. In *Proceedings of the Electronic Components and Technology Conference*, 2004.

[24] K. Xiu and M. Ketchen. Generalized thermal analysis of hotspots on a high power density microprocessor chip. In *Proceedings of the Electronic Components and Technology Conference*, 2005.

[25] Y. Zhan and S.S. Sapatnekar. Fast computation of the temperature distribution in VLSI chips using the discrete cosine transform and table look-up. In *Proceedings of the Asia/South Pacific Design Automation Conference*, 2005.

[26] Y. Zhan and S.S. Sapatnekar. A high efficiency full-chip thermal simulation algorithm. In *Proceedings of the International Conference on Computer-Aided Design*, 2005.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803